계층형 명세 언어의 정의 및 검사 (Definition and Verification of Hierarchical Specification Language)

박 사 천, 권 기 현
경기대학교 정보과학부
{sachem, khkwon}@kyonggi.ac.kr

요 약

시스템 정형검증의 한 분야로 모형검사 방법이 활발히 연구되어 왔다. 왜냐하면 모형검사 방법은 사용자의 개입 없이 자동적으로 검증을 할 수 있기 때문이다. 그러나 모형검사에는 몇 가지 문제점이 있다. 그 중 가장 심각한 것이 상태폭발 문제이다. 상태폭발로 인해서 모형검사기는 큰 규모의 시스템들을 다루지 못한다. 시스템은 명세 단계에서 계층적으로 명세 되는 반면 검증 단계에서는 계층성을 잃고 평탄화 된다. 이렇게 평탄화 될 경우 전이의 수와 상태의 수가 폭발적으로 늘어나게 된다. 본 연구는 평탄화에 의해 발생하는 상태폭발의 문제를 극복하기 위하여, 계층형 크립키 구조를 정의하고 그 구조에서 CTL 속성을 검사하는 모형검사 알고리즘을 제시한다.

1. 서론

하드웨어와 소프트웨어 기술이 발달하고 또 사회 전반적으로 보급됨으로 인해서 시스템에 대한 요구가 많아졌고 의존도 또한 높아졌다. 시스템이 커지고 복잡해짐에 따라 시스템의 미묘한 오류를 발견하기는 더욱 어렵게 되었고 그 피해가 속출하였다. 그러므로 시스템의 정확성을 검증하는 작업은 개발과정에서 필수적인 사항이 되었다. 비단 안전성이 특히 요구되는 군사분야 혹은 최첨단 산업분야 뿐 아니라 상업적인 분야나 가전제품에 들어가는 내장형 시스템에 이르기까지 일상적인 분야에 사용되는 시스템들에도 정확성은 중요한 문제가 되었다.

정확성을 검사하는 기존의 시험기법과 모의시험 등과는 상호 보완적인 방법으로 최근 20여년간 정형방법이 활발히 연구되어 왔다. 정형방법(Formal Methods)은 수학을 기반으로 하여 검색할 시스템을 철저히 분석하는 기법인데 크게 정형명세(Formal Specification)와 정형검증(Formal Verification)으로 나뉜다. 정

* 본 연구는 전자통신연구원 위탁 과제(3010-2002-0090) 연구비 지원으로 수행되었음.

형검증은 다시 증명을 기반으로 하는 정리증명(Theorem Proving)과 모형을 기반으로 하는 모형검사(Model Checking) 방법으로 나뉜다. 정리증명은 시스템의 모든 행위를 검사하지만 전문가의 도움을 필요로 한다는 단점이 있고 모형검사는 시스템의 개별 속성만을 검사하지만 완전 자동이라는 이점이 있다. 80년대 초반에 E. Clarke에 의해서 CTL(Computation Tree Logic)모형검사 방법이 소개된 이후 실험적으로 연구되다가 87년 McMillan이 OBDD(Ordered Binary Decision Diagram)를 이용한 SMV(Symbolic Model Verifier)를 개발하여서 산업현장에서 모형검사기법에 대한 적용가능성을 열어주었다[1]. 실제로 하드웨어와 프로토콜 분야에서 이전에 발견하지 못했던 중요한 오류들을 수 차례 발견하였고 최근에는 모형검사를 이용해서 항공기 소프트웨어 시스템의 오류를 발견하게 되었다. 그러나 CTL모형검사는 본질적으로 평탄한 구조에서 해석되기 때문에 실제의 커다란 시스템에 적용하기에는 아직 많은 문제점을 안고 있다. 먼저 무한한 시스템의 행위를 유한한 상태 기계로 모형화 하는 것이 어렵고, 이렇게 모형화가 되었다 하더라도 검색해야 할 상태공간이 처리할 수 없이 커지는 상태폭발 문제(state explosion problem)가 발생한다. 상태폭발을 극복하기 위한 연구로는 합성을 이용하는 방법, 추상화를 이용하는 것, 대칭성과 반 순서를 이용하는 것 등이 있다. 또 다른 방향의 연구로는 계층성을 이용한 방법인데 이것은 계층적으로 설계된 시스템의 평탄화를 피하고 시스템의 계층성을 이용해서 탐색해야 할 상태공간을 효율적으로 검사하는 방법이다[2].

본 논문에서는 계층성을 이용한 방법을 제안한다. 논문의 순서는 2장에서 시스템의 모형인 계층형 크립키 구조(Kripke Structure)를 정의하고 3장에서 CTL식의 일반적인 정의를 살펴본 후에 4장에서 계층형 크립키 구조에서 정의된 CTL식이 어떻게 해석되는지 그 알고리즘을 보인다. 마지막으로 결론과 향후 연구를 제시한다. 우리가 제시한 계층형 크립키 구조는 계층간의 전이(Transition)를 허용하지 않으며 상위 구조에서 기술된 속성(Property)이 하위 구조로 상속되는 특성이 있다. 속성 상속은 동일한 특성을 지니는 상태들을 묶어서 계층으로 표현한 상태도(Statecharts)의 기본원리를 응용한 것이다[3]. 본 논문에서 제시하는 계층은 단순한 크립키 구조의 중첩으로 표현했고 이것은 실세계를 단순하고 명확하게 표현할 수 있다. 본래 상태도에는 전이가 계층을 넘나들며 발생한다. 그리고 이것은 시스템을 명세 하는데 유용하게 이용된다. 하지만 Argo[4]나 모드[5]같은 구조에서는 계층간의 전이를 허용하지 않는다. 프로그램이 명확하게 서브프로그램으로 분해되기 위해서는 계층간의 전이는 억제되어야 하기 때문이다. 또한 [5]에서는 계층을 모드라는 핵심 컴포넌트의 중첩으로 표현했는데 이때 서브 모드를 블랙박스처럼 여기고 외부에는 입(출력 전이만을 허용하고 있다. 모듈성을 지닌 프로그램에서 이러한 형태를 쉽게 발견할 수 있으며 컴포넌트의 결합으로 된 시스템의 모형검사에는 이러한 모형은 더욱 바람직할 것이다. 본 논문에서는 계층을 뛰어 넘는 전이(Inter-level transition)를 허용하지 않음으로써 보다 소프트웨어 시스템에 가까운 모형의 성격을 부여했다. HSEM(Hierarchical State/Event Model)에서는 병행적인 요소를 다루고 있지만 일반적인 시

제속성이 아닌 도달성 검사에 국한한 연구였다[6]. 본 논문에서는 순차적인 구조에서 CTL속성을 검사한다. 병행적인 구조에 대한 검사는 더 연구되어야 할 과제이다.

2. 계층형 크립키 구조

계층형 크립키 구조는 앞에서 언급한 바와 같이 크립키 구조의 중첩으로 이루어져 있다. 크립키 구조는 양상논리의 의미를 정의하기위해 1963년 크립키에 의해서 제시된 상태 전이도 이다. 이것은 시스템의 행위를 유한상태로 모형화 하는데 널리 사용된다. 원래의 크립키 구조는 상태와 전이 그리고 시작 상태들의 집합으로 정의 되지만 시스템의 시작은 동일한 한 상태로부터라고 가정하면 하나의 시작상태로 정의할 수 있다.

정의 1. 크립키 구조 M = (S, in, R)은 세 개의 항목으로 이루어지며 아래와 같이 정의된다.
- S: M에 속하는 모든 상태들의 집합
- in ∈ S: M의 시작 상태
- R: 전이관계(transition relation), $\forall s \in S. \exists t \in S.$ (s, t)∈R이고 전체관계(total relation)이다.

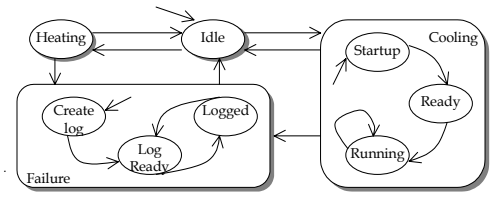
계층형 크립키 구조는 크립키 구조의 상태 안에 새로운 크립키 구조가 정의되는 형태를 취한다. 따라서 n개의 크립키 구조와 이들의 포함관계 즉, 계층관계로 이루어진다. 크립키 구조의 집합을 H = {M_1, M_2, …, M_n}라 하고 H에 속한 각각의 크립키 구조 M_i = (S_i, in_i, R_i)이 된다. 이때 모든 계층에 포함된 상태들을 $\sum = \bigcup S_i$ (i=1 .. n) 로 정의하면 계층 관계를 나타내는 함수는 $\Upsilon: \sum \rightarrow 2^{\sum}$로 정의된다. 함수 Υ는 상태 s∈∑를 상태집합 ∑′⊂∑으로 사상 시킨다. 이때 ∑′은 s의 자식 상태들의 집합이다. $\Upsilon^+(s)$는 s자신을 포함하지 않는 s의 모든 자손 상태들의 집합이고, $\Upsilon^*(s) = \Upsilon^+(s) \cup \{s\}$이다.

정의 2. 계층형 크립키 구조 HK = (N, H, Υ, L)는 네 개의 항목으로 이루어지며 N∈H은 최상위 크립키 구조이고 라벨함수 $L: \sum \rightarrow 2^{AP}$는 임의의 상태 s를 기본 명제의 집합 AP의 부분 집합으로 사상한다.

전이관계 $R_i \subseteq S_i \times S_i$는 같은 i-계층의 상태들 간에 정의 되기 때문에 모형에서 계층간의 전이는 존재하지 않는다. 각 상태에 고유한 속성을 표현하는 라벨함수(Labeling function) L은 모든 상태에 라벨(Label) 할 수 있음을 보여준다. 이것은 속성 상속을 간단히 표현한다. 예를 들어 라벨함수에 의해서 상태 s에 기본명제 p가 라벨 되었다면 상태s를 포함한 모든 자손 상태 $\Upsilon^*(s)$에도 동일하게 라벨 된다고 간주한다. 전이가 발생했을 때 도착 상태가 크립키 구조를 포함하는 상태일 경우 다음 상태는 그 크립키 구조의 상태들 중 시작 상태가 된다. 어떤 상태 안에 포함된 연속된 시작 상태들은 평탄화된 구조에서 하나의 상태와 같다. 따라서 계층형 구조에서 CTL 알고리즘을 적용하기 위해서 이러한 상태들을 표현할 수 있는 함수가 필요하다.

정의 3. In이 HK에 포함된 모든 시작 상태들의 집합이라고 하면 어떤 상태 s가 포함하는 크립키 구조의 시작상태를 구하는 함수 I와 시작상태의 연속을 구하는 함수 I^+는 각

각 다음과 같이 정의된다: $I(\{s\}) = \Upsilon(s) \cap In$, $I^{+}(\{s\}) = I(\{s\}) \cup I(I(\{s\})) \cup \cdots$, $I^{*}(\{s\}) = I^{+}(\{s\}) \cup \{s\}$.

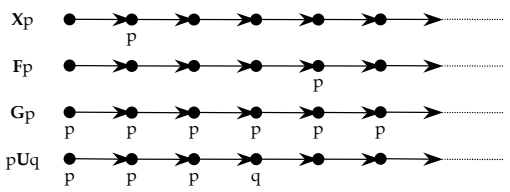


(그림 1) 자동 온도 조절기 모형

$I^{*}(\{s\})$는 상태 s와 s에 포함된 연속된 시작 상태들의 집합을 의미하는 것인데 이것은 평탄화 된 구조에서 하나의 상태로 나타난다. S_M이 임의의 크립키 구조 M에 대한 상태 집합이라 할 때 어떤 상태s가 크립키 구조 M을 포함하고 있을 때 $\Upsilon(s)$는 S_M와 동일하다. [그림 1]은 계층형 크립키 구조를 설명하기 위해 자동 온도 조절기를 간단히 나타낸 것이다. 최상위 크립키 구조 N의 상태집합은 S_N = {Heating, Idle, Cooling, Failure}이다. N계층의 초기 상태는 Idle이다. N계층의 전이 관계는 S_N에 포함된 상태들의 전이 관계만으로 이루어진다. N계층에 포함된 상태 Failure에 포함된 크립키 구조를 $M_{Failure}$라고 하고 상태 Cooling에 포함된 상태를 $M_{Cooling}$이라 한다면 최상위 계층의 크립키 구조를 M_{Root}=N라고 할 때 H = {M_{Root}, $M_{Failure}$, $M_{Cooling}$}이다. 이때 Σ = {Heating, Idle, Cooling, Failure, Startup, Ready, Running, Create, LogReady, Logged} 이다. 온도 높아져서 Idel상태에서 Cooling상태로 전이가 발생하면 시작상태를 구하는 함수 I({Cooling}) = {Startup}에 의해서 Startup상태로 전이된다. 계층함수 Υ에 의해서 Cooling의 하위 상태들을 얻을 수 있다: Υ(Cooling) = {Startup, Ready, Running}. 기본명제의 집합 AP = {normal, tooCool, tooHot}라고 가정할 때, 라벨링 함수에 의해 L(Idle) = {Normal}, L(Heating) = {tooCool}, L(Cooling) = {tooHot}이라고 하면 Υ^{*}(Cooling)인 상태에 모두 tooHot이라는 속성이 만족되는 것이다. 즉 Cooling상태에 라벨된 속성은 그 하위에 포함된 모든 상태에서도 라벨 된 것으로 간주한다.

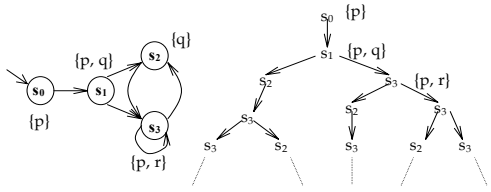
3. CTL 명세언어

시스템을 유한 상태 모형으로 명세할 때는 시스템이 만족해야 할 속성을 주로 시스템의 행위를 시간의 전후 관계로서 묘사하는 시제논리로 명세 한다. 시제논리에는 선형 시제논리와 분기 시제논리가 있다. 대표적인 선형시제 논리로는 PLTL(Propositional Linear Temporal Logic)이 있고 대표적인 분기시제 논리로는 CTL이 있는데 CTL은 크립키 구조를 무한 트리의 관점에서 해석한 것이다. CTL은 기본적으로 명제논리에 기반하고 거기에 경로 한정자(path quantifier)와 시제 연산자(temporal operator)가 추가되어 확장된 논리이다. 네 개의 시제 연산자(neXt, Future, Global, Until)는 모두 현재 상태를 기준으로 해석된다. [그림 2]에서와 같이 각 시제연산자는 상태의 무한 연속으로 된 경로상에서 해석되는데, Xp는 “다음 상태에서 p가 만족된다.”를 의미하고 Fp는 “언젠가는 p를 만족한다.”, Gp는 “모든 상태에서 p가 만족된다.”는 의미로 해석된다. pUq는 “q가 만족 될 때 까지 p가 만족된다.”는 의미이다.



(그림 2) 시제연산자 해석의 예

크립키 구조를 [그림 3]에서와 같이 시작 상태로부터 연속된 상태들의 트리로 표현할 수 있다.두개의 경로 한정자 A, E는 각각 “모든 경로”와 “어떤 경로”로 해석된다. 따라서 두 개의 경로 한정자와 네 개의 시제연산자를 결합하여 8개의 연산자들(AX, AF, AG, AU, EX, EF, EG, EU)을 만들 수 있다. 예를 들어 EGp는 “현재 상태로부터 모든 상태에서 p를 만족하는 경로가 적어도 하나 존재한다.”는 의미이다. [그림 3]의 오른쪽 계산 트리(Computation Tree)에서 이 식을 해석하면 경로 π = $s_0s_1s_3s_3s_3$…의 모든 상태에서 속성 p가 만족하므로 EGp도 만족한다. AFp는 “모든 경로에서 언젠가는 p를 만족한다.”는 의미가 되고 아래의 그림 3에서는 시작상태 s_0가 p로 라벨 되어 있기 때문에 이 식 또한 만족한다. 정의 4와 정의 5는 CTL 논리의 구문과 의미의 정형적인 정의이다[7].



(그림 3) 크립키 구조와 계산 트리

정의 4. CTL구문을 BNF형식으로 나타내면 다음과 같다:

$\phi := p \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid EX\phi \mid EG\phi \mid E[\phi_1 U \phi_2]$

위 정의의 CTL구문에서는 ¬, ∧, EX, EG, EU만을 보였다. CTL식의 다른 구문은 동치 관계를 이용해서 모두 표현될 수 있다: $\phi_1 \vee \phi_2 \equiv \neg\phi_1 \wedge \neg\phi_2$, $\phi_1 \Rightarrow \phi_2 \equiv \neg\phi_1 \vee \phi_2$, $\phi_1 \Leftrightarrow \phi_2 \equiv \phi_1 \Rightarrow \phi_2 \wedge \phi_2 \Rightarrow \phi_1$, $A\pi \equiv \neg E \neg \pi$, $F\phi \equiv true U \phi$, $G\phi \equiv \neg F \neg \phi$.

정의 5. CTL의 의미(semantics)는 크립키 구조 M과 기본명제 집합 AP에서 아래와 같이 귀납적으로 정의된다:

$M,s_0 \models p$ iff $p \in L(s_0)$

$M,s_0 \models \neg\phi$ iff $M,s_0 \not\models \phi$

$M,s_0 \models \phi_1 \wedge \phi_2$ iff $M,s_0 \models \phi_1$ and $M,s_0 \models \phi_2$

$M,s_0 \models EX\phi$ iff $\exists t.(s_0,t) \in R$ and $M,t \models \phi$

$M,s_0 \models EG\phi$ iff $\exists \pi = s_0,s_1,s_2,\cdots, \forall i, i \geq 0,$ $M,s_i \models \phi$

$M,s_0 \models E[\phi 1 U \phi 2]$ iff $\exists \pi = s_0,s_1,s_2,\cdots, \exists k \geq 0,$ $M,s_k \models \phi_2 \wedge \forall i.0 \leq i < k,\ M,s_i \models \phi_1$

$p \in AP$는 기본명제 이고 π는 M에서 상태들의 무한 연속인 경로이다: $\pi = s_0, s_1,\cdots, i \geq 0, (s_i, s_{i+1}) \in R$. 위의 정의에서 정의하지 않은CTL식 AXϕ, EFϕ, AFϕ, AGϕ, A[$\phi_1 U \phi_2$]등에 대한 의미는 정의 5를 이용하여 유도 될 수 있다.

4. 모형검사 알고리즘

아래의 [그림 4]는 계층형 크립키 구조에서의 CTL모형검사 알고리즘을 대략적으로 보인 것이다. 알고리즘은 계층형 크립키 구조 HK와 CTL식 ϕ를 입력 받아서 HK에 포함된 모든 상태들 중에 ϕ를 만족하는 상태들의 집합을 돌려준다. 일반적인 CTL모형검사 알고리즘은 Labeling 알고리즘이다. 상태들

의 무한연속(infinite sequence)인 계산 트리를 놓고 해석한다면 편하겠지만 시스템에서 그것을 구현하기가 불가능 하기 때문에 식이 만족되는 상태들에 그 식을 라벨하는 것으로 구현한다. 알고리즘은 식을 분해해서 제일 길이가 작은 부분식부터 재귀적으로 호출해서 전체 길이의 식까지 만족되는 상태들의 집합을 돌려준다.

[그림 4]의 알고리즘은 기존의 CTL모형검사 알고리즘을 계층성에 맞게 확장한 것이다. true는 전체 상태인 Σ를 돌려주고 flase는 공집합을 돌려준다. AP에 대한 라벨은 이미 각 상태에 라벨 되어 있으므로 라벨 된 상태와 그 상태가 포함하는 모든 크립키 구조의 상태들의 집합을 돌려준다. 명제논리식 ¬ϕ에 대한 계산은 $\{s:\Sigma|\Upsilon^*(s)\cap HSat(\phi)\neq\varnothing\}$가 되는데 이것은 상태 자신뿐만 아니라 자신이 포함하는 모든 상태들에서도 ϕ가 나타나지 않는 상태들의 집합을 의미한다. 또 다른 명제논리식 ϕ∧Ψ은 간단히 ϕ와 Ψ가 만족하는 상태 집합의 교집합으로 구한다. 시제논리식 EXϕ는 전이관계 (s, s′)∈R에 속한 다음 상태 s′자신과 s′에 포함된 연속된 시작상태에서 ϕ가 만족하면 상태 s와 s에 포함된 모든 상태를 돌려준다. EX까지는 쉽게 구할 수 있다. EG와 EU는 또 다른 서브루틴을 호출한다.

```
function HSat(HK, ϕ) : set of State;
begin
  if ϕ = true ⟶ return Σ
  []ϕ = false ⟶ return ∅
  []ϕ ∈ AP ⟶ return {s:Σ | ∃s′∈Σ.ϕ∈L(s′) ∧∀s∈γ′(s′) }
  []ϕ = ¬φ ⟶ return {s:Σ | γ′(s) ∩ HSat(φ) ≠ ∅}
  []ϕ = φ ∧ ψ ⟶ return (HSat(φ) ∩ HSat(ψ))
  []ϕ = EXφ ⟶ return {t:Σ | ∃s∈Σ.(s, s′)∈R∧I*(s′)
                          ∩ HSat(φ) ≠ ∅.∀t∈γ′(s)}
  []ϕ = EGφ ⟶ return HSat_EG(S_N, φ)
  []ϕ = E[φUψ] ⟶ return HSat_EU(S_N, φ, ψ)
  fi
end
```

Model Checking problem: $I^*(\{in_N\})\cap HSat(\phi)\neq\varnothing$.

(그림 4) 계층형 크립키 구조의 모형검사 알고리즘

계층형 크립키 모형에서 CTL 모형검사의 만족성(HK⊨ϕ) 문제는 $I^*(\{in_N\})\cap HSat(\phi)\neq\varnothing$가 참이 될 때 만족하는 것으로 판명된다. 먼저 CTL식으로 표현된 속성 ϕ를 만족하는 상태들의 집합을 계산한다. 그리고 최상위 크립키 구조 N의 시작상태와 그 상태에 포함된 연속된 시작 상태들의 집합과 ϕ를 만족하는 상태집합을 교집합 한다. 교집합이 공집합인지를 확인 한다. 앞서 언급한 바와 같이 $I^*(\{in_N\})$은 평탄화 했을 경우 하나의 시작상태가 되기 때문에 속성 ϕ가 루트계층의 시작상태 연속 중에 라벨 되어 있으면 계층형 크립키 구조 HK에서 속성 ϕ가 만족하는 것이다.

4.1. EG와 EU의 서브 알고리즘

[그림 5]는 EGϕ를 만족하는 상태집합을 구하는 알고리즘이다. 계층성의 이점을 살리기 위해 상위 계층의 전이를 최대한 활용하도록 하고 있다. 알고리즘을 시작할 때는 최상위 계층에 해당하는 크립키 구조의 상태집

합을 대상으로 하고 재귀적으로 하위 구조들을 검사하게 된다. 중복 검사를 피하기 위해 먼저 검사된 상태는 다음 번 반복에서 제외된다.

```
function HSat_EG(S_M: set of State, φ: Formula) : set of State;
begin var Q, Q', Sub, V, Z, Z': set of State;
  Q, Q' := S_M ∩ HSat(φ), S_M;
  do Q ≠ Q' ⟶
    Q' := Q;
    Q := (Q∩{s | ∃s'∈Q.(s, s')∈R}) ∪ S_φ
  od;
  foreach s ∈ (S_M/Q)
    if γ(s) ≠ ∅ ⟶
      Sub := HSat_EG(γ(s), φ);
      Z, V := ∅, ∅;
      if Sub ∩ I⁺({s}) ≠ ∅ ⟶
        Z, Z' := {s}, ∅;
          do Z ≠ Z' ⟶
            Z' := Z;
            V := V ∪ ({t | ∃t'∈Z.(t, t')∈R}/[φ]);
            Z := Z ∪ ({t | ∃t'∈Z.(t, t')∈R} ∩[φ])
          od;
        Z := (Z/{s}) ∪ V
      fi;
      Q := Q ∪ Z ∪ Sub
    fi;
  return {s:Σ | ∃s'∈Q.s∈ (γ*(s') ∩ HSat(φ))}
end
```

(그림 5) 서브루틴 $HSat_{EG}(S_M, \phi)$, EGϕ를 만족하는 상태집합을 구할 때 두 번의 고정점 계산 알고리즘을 사용한다. I_ϕ = {s:S_M|I^+({s}) ∩ HSat(ϕ)≠∅}는 상태 s에 포함된 시작 상태의 연속 중에 ϕ를 만족하는 상태가 존재하는 상태집합 이고, 상태 s에 포함된 구조들의 모든 상태들에서 ϕ를 만족하는 상태집합을 S_ϕ={s:S_M|s∈HSat(ϕ)∧Υ^+(s)≠∅}라고 표현했다. 식을 간단히 표현하기 위해 [ϕ]을 I_ϕ ∪HSat(ϕ)으로 정의했다

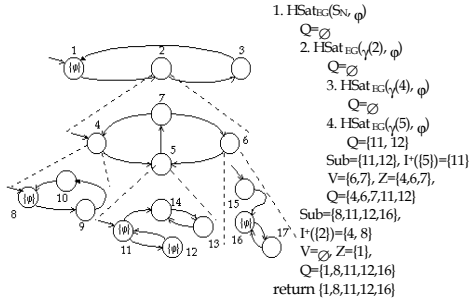
첫 번째 계산 루틴은 입력 받은 상태집합 즉, 하나의 크립키 구조에서 전통적인 CTL 모형검사의 EGϕ와 동일하게 최대 고정점 계산으로 이루어진다. 만일 입력 받은 상태 집합에서 EGϕ가 만족되는 상태를 제외한 나머지 상태들 중 다른 크립키 구조를 포함하는 상태가 있으면 그 상태를 대상으로 재귀적으로 알고리즘을 수행한다. 깊이 우선 방식으로 최하위 계층까지 수행한 후 만족되는 집합이 있을 때 다시 거꾸로 상위 구조들과의 관계를 최소 고정점으로 계산하게 된다. 계층을 내려가면서 EGϕ를 찾고 다시 올라오면서 만족되는 상태들을 찾는다. [그림 6]은 E[ϕUΨ]를 계산하는 알고리즘이다. EGϕ를 구하는 알고리즘과 거의 흡사하다. 다만 최소 고정점을 두 번 사용하는 것이 다른데 이것은 EU의 계산이 최소 고정점 계산이고 하위 계층으로부터 상위로 만족되는 상태들을 걷어 올리는 부분 또한 최소 고정점으로 계산 되기 때문이다.

```
function HSat_EU(S_M: set of State, φ, ψ: Formula) : set of State;
begin var Q, Q', Sub, V, Z, Z': set of State;
  Q, Q' := S_M ∩ HSat(ψ), ∅;
  do Q ≠ Q' ⟶
    Q' := Q;
    Q := Q∪({s | ∃s'∈Q.(s, s')∈R} ∩ [φ])
  od;
  foreach s ∈ (S_M/Q)
    if γ(s) ≠ ∅ ⟶
      Sub := HSat_EU(γ(s), φ, ψ);
      Z, V := ∅, ∅;
      if Sub ∩ I⁺({s}) ≠ ∅ ⟶
        Z, Z' := {s}, ∅;
          do Z ≠ Z' ⟶
            Z' := Z;
            V := V ∪ ({t | ∃t'∈Z.(t, t')∈R}/[φ]);
            Z := Z ∪ (({t | ∃t'∈Z.(t, t')∈R} ∩ [φ])/Q)
          od;
        Z := (Z/{s}) ∪ V
      fi;
      Q := Q ∪ Z ∪ Sub
    fi;
  return {s:Σ | ∃s'∈Q.s∈ (γ*(s') ∩ (HSat(φ) ∪ HSat(ψ)))}
end
```

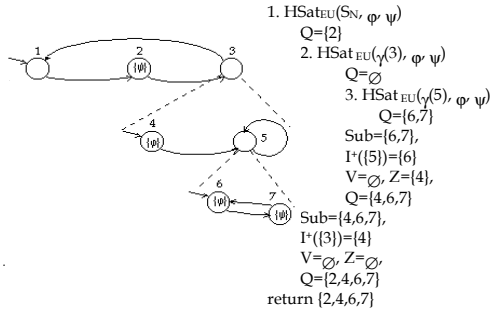
(그림 6) E[ϕUΨ]를 검사하는 알고리즘

[그림 7]은 EGϕ 알고리즘의 적용 예이다. 이 예는 계층구조를 탐색하는 방법을 보이기

위해 만든 예제이다. 따라서 만일 [그림 7]의 3번 상태에 ϕ가 라벨 되었다고 가정하면 단 한번의 $HSat_{EG}(S_N, \phi)$ 호출로 EGϕ에 대한 만족성 검사가 완료될 것이다. 다시 말해 상위 계층의 상태에 속성이 라벨 된다면 계산 과정은 더 짧아질 것이다. 아래의 경우 식이 만족되는 집합에 루트 크립키 구조의 시작 상태가 포함되므로 이 모형에서 EGϕ가 만족한다. [그림 8]은 E[ϕUΨ]를 계산하는 알고리즘의 사례를 보인 것이다. 여기서 4, 6번 상태에 ϕ가 라벨 되고 2, 7번 상태에 Ψ가 라벨 된 경우에 E[ϕUΨ]를 검사하는 예이다. 초기 상태에 ϕ가 라벨 되지 않기 때문에 식이 만족하지 않는다.



(그림 7) EGϕ 알고리즘의 적용 예



(그림 8) E[ϕUΨ] 알고리즘의 적용 예

4.2. 간단한 예제

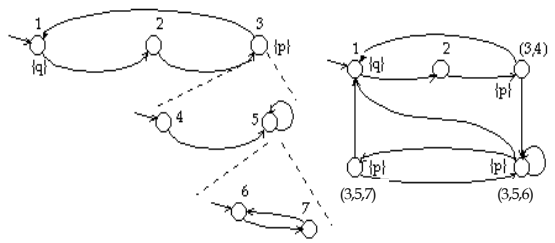
AP={p, q}이고 주어진 계층형 크립키 구조가 [그림 9]의 왼쪽과 같을 때 AG(p ⇒ AFq)의 속성이 만족하는가를 검사해보자. 먼저 시제 연산자들의 동치관계에 의해서 다음과 같은 식을 얻는다:

$$\neg E[true\ U\ (p \wedge EG\neg q)]$$

그 다음에는 식을 아래와 같이 부분식으로 분해하고 부분식의 길이가 가장 작은 AP 속한 것으로부터 라벨링 알고리즘이 재귀적으로 적용되면서 전체 식을 만족하는 상태집합을 얻는다:

$$\begin{aligned}
&Sub\text{-}formula(\neg E[trueU(p \wedge EG\neg q)]) \\
&= \{(\ E[true\ U\ (p \wedge EG\neg q)]\} \\
&\cup \{E[true\ U\ (p \wedge EG\neg q)]\} \\
&\cup \{true\} \cup \{\ p \wedge EG\neg q\} \\
&\cup \{p\} \cup \{EG\neg q\} \cup \{(q\} \cup \{q\}
\end{aligned}$$

예를 들어 HSat(EG¬q)를 계산하는 단계에서는 알고리즘에 의해 S_N계층만을 검사하면 된다. 그러나 [그림 9]의 오른쪽과 같이 평탄화 된 구조에서 동일한 식을 계산하려면 전체 전이관계와 상태들을 모두 고려해야 한다. 평탄화 된 구조에서 3개로 표현된 전이들(3,4→1, 3,5,6→1, 3,5,7→1)이 계층형 구조에서는 하나의 전이(3→1)로 줄어 들고 기본 명제에 대한 라벨도 상위상태에 되어있기 때문에 효과적인 모형검사를 수행할 수 있다. 이러한 이점은 라벨을 진행하면서도 얻을 수 있다.



(그림 9) 계층형과 평탄화 된 구조의 속성검사 비교

[그림 9]에서는 주어진 식 ¬E[true U (p ∧ EG¬q)]로 라벨 된 상태가 없기 때문에 1)의 계층형 구조에서 주어진 속성이 만족하지 않음을 알 수 있다. 이 알고리즘은 각각의 부분식에 대해 아래 과정으로 수행된다.

HSat(q) = {1}
HSat(¬q) = {2,3,4,5,6,7}
HSat(EG¬q)=$HSat_{EG}(S_N, ¬q)$={2,3,4,5,6,7}
HSat(p) ={3,4,5,6,7}
HSat(EG¬q) ∩ HSat(p) ={3,4,5,6,7}
HSat(E[true U (p∧EG¬q)] = $HSat_{EU}(S_N$, true, p∧EG¬q) = Σ
HSat(¬E[trueU(p∧EG¬q)]) = ∅

알고리즘의 수행도중에 검사할 필요가 없는 하위구조는 피해갈 수 있다. 예를 들어서 EXϕ를 검사하는 것에 대해 고려해보자. 먼저 ϕ가 만족되는 상태들을 검사한 후 ϕ를 만족하는 상태들의 이전 상태에 EXϕ를 라벨함으로써 그 하위에 포함된 모든 상태들에 대해서도 동일하게 라벨함과 같은 의미를 얻을 수 있다.

5 결론 및 향후 연구

본 논문에서는 계층성을 보다 적극적으로 이용하여 모형검사의 문제점인 상태폭발에 대응하였다. 계층형 크립키 구조를 정의했고 계층형 구조 상에서 해석되는 CTL알고리즘은 제시했다. 계층형 크립키 구조에서 상위 상태에 라벨 된다는 것을 그것이 포함하고 있는 하위의 모든 상태에 라벨 된다는 의미로 정의했다. 이로써 최상위 계층으로부터 모형검사를 시작하여서 하위상태로 내려가면서 더 이상 하위를 검사할 필요가 없는 것들은 생략함으로 모형검사의 처리과정을 개선시켰다.

하지만 우리가 제시한 모형은 실제 산업에서 사용되는 상태도와 같은 명세언어와는 많은 차이를 보인다. 그 중에서도 계층형 모형에 병행적인 요소를 표현할 수 없는 단점이 있었다. 그러나 병행성은 시스템 명세에 있어 중요한 특성이기 때문에 앞으로의 연구에서는 모형에 병행성을 추가하는 연구가 이루어져야 할 것이다. 여기에는 두 가지 방향이 있다. 먼저 생각해 볼 수 있는 것이 기존의 병행성을 가진 계층형 명세 언어들을 본 논문에서 제시한 계층형 크립키 구조로 변환하여 처리하는 것과 계층형 크립키 구조 내에 직접 병행성을 추가하는 것이 있다. 또한 본 논문에서 제시한 알고리즘의 정확성과 복잡도에 관한 연구와 이를 바탕으로 계층형 CTL모형검사 도구를 구현하는 과제가 남았다.

참고문헌

[1] E. Clarke, D. Long. “Verification tools for finite-state concurrent systems”, in Proceedings of A Decade of Concurrency, Lecture Notes in Computer Science 803, pp.124-175, 1994.

[2] R. Alur, M. Yannakakis, “Model checking of hierarchical state machines”, ACM TOPLAS, Vol.23, No.3, 2001.

[3] D. Harel, A. Naamad, “The STATEMATE semantics of statecharts”, ACM Transactions on Software Engineering and Methodology, Vol.5, No.4, pp.293-333, 1996.

[4] F. Maraninchi, “Operational and compositional semantics of synchronous automaton compositions”, in Proceedings of CONCUR'92, Lecture Notes in Computer Science 630, pp.550-564, 1992.

[5] M. McDougall, R. Alur, R. Grosu, “Efficient reachability analysis of hierarchical reactive machines,” in Proceedings CAV'00, Lecture Notes in Computer Science 1885, 2000.

[6] G. Behrmann, et.al., “Verification of hierarchical state/event systems using reusability and compositionality,” in Proceedings of TACAS'99, 1999.

[7] J.P. Katoen, “Concepts, algorithms, and tools for model checking”, Arbeitsberichte der IMMD, Vol.32, No.1, Friedrich-Alexander-Universitat Erlangen-Nurnberg, 1999.

박사천

1993년~2001년 경기대학교 정보과학부(학사).

2001년~현재 경기대학교 전자계산학과 석사과정 중.

관심분야는 소프트웨어 공학, 정형방법 등.

권기현

1981년~1985년 경기대학교 전자계산학과(학사).

1985년~1987년 중앙대학교 전자계산학과(이학석사).

1987년~1991년 중앙대학교 전자계산학과(공학박사).

1991년~현재 경기대학교 정보과학부 교수.

1998년~1999년 독일 드레스덴공대학 전산과 방문교수.

1999년~2000년 미국 카네기멜론대학 전산과 방문교수.

관심분야는 소프트웨어 공학, 정형방법 등.